# Pushing data into CP models using Graphical Model Learning and Solving

Céline Brouard, Simon de Givry, and Thomas Schiex

Université Fédérale de Toulouse, ANITI, INRAE, UR 875, Toulouse, France
{firstname.name@inrae.fr}

**Abstract.** Integrating machine learning with automated reasoning is one of the major goals of modern AI systems. In this paper, we propose a non-fully-differentiable architecture that is able to extract preferences from data and push it into (weighted) Constraint Networks (aka Cost Function Networks or CFN) by learning cost functions. Our approach combines a (scalable) convex optimization approach with empirical hyper-parameter tuning to learn cost functions from a list of high-quality solutions. The proposed architecture has the ability to learn from noisy solutions and its output is just a CFN model. This model can be analyzed, empirically hardened, completed with side-constraints and directly fed to a Weighted Constraint Satisfaction Problem solver.
To explore the performances and range of applicability of this architecture, we compare it with two recent neural-net friendly learning systems designed to "learn to reason" on the Sudoku problem and also show how it can be used to learn and integrate preferences into an existing CP model, in the context of Configuration systems.

**Keywords:** Graphical Models · Cost Function Networks · Learning · Constraint Programming.

## 1 Introduction

Constraint Satisfaction and Constraint Programming define a powerful framework for modeling and solving decision problems. It is often considered as one of the closest approaches computer science has made to the Holy Grail of programming: "the user states the problem, the computer solves it." [16]. The problem may however be difficult to state, not only because of the rich CP language, but because several of the aspects of the real problem may be inaccessible to the modeler, leading to approximate formulations, providing only partially satisfactory solutions. In this paper, we show how preferences and constraints can be extracted from historical solutions so that they can be directly represented inside a (weighted) constraint satisfaction problem.

[0] This is an edited and improved version of the published version of CP2020 without the "hint as images/solutions as number" situation that is not a realistic situation. On the visual Sudoku task, SAT-Net (and us now) only present results in the situation where both hints and solutions are available as images).

In this paper, we are interested in learning a criterion and its domain of definition as a set of cost functions and constraints, starting from a set of good-quality solutions that could require a perceptive layer for acquisition. The learned preferences and constraints are represented as a Cost Function Network (CFN). This learned CFN can then be completed by user-defined constraints or criteria before feeding a Weighted Constraint Satisfaction Problem (WCSP) solver. Such a workflow is very useful when the aim is to produce a new solution that resides in a large family of known designs [34] (providing data), that must satisfy both known general requirements and new specific properties.

Our main contribution is to leverage the capacity of CFN solvers to optimize Graphical Models (GMs), a family of models that covers Constraint Networks, Clausal Propositional Logic and their weighted variants as well as probabilistic Markov Random Fields and Bayesian Nets models [10]. Starting from historical solutions, we use a recent convex optimization approach to estimate a CFN model of the data that gives a lower cost to the training set. We notice that a maximum regularized approximate log-likelihood loss [30] does tackle this objective. We use a scalable algorithm which learns the scopes and cost tables of cost functions. This CFN can then be optionally enriched by user cost functions or constraints and solved by a WCSP solver for various inputs. The resulting architecture combines ML and CP components in an way which, in our knowledge, has never been tested to learn preferences (and constraints).

Our approach compares with recent differentiable "learning to reason" architectures such as Recurrent Relational Nets (RRN [29]) or SAT-Net [37] in terms of input, output and prior information (assumptions). These approaches define fully differentiable layers that can learn pairwise "message passing" functions (RRN) or a low-rank convex relaxation of Max-SAT, using continuous descent algorithms as their optimization component. Such layers are easy to inter-operate with Deep learning differentiable architectures. These two approaches have been benchmarked on Sudoku resolution. We therefore compare our approach to the RRN and SAT-net approaches, including in situations where Sudoku grids are only available as hand-written grid images. We observe that our ML+CP approach offers a better accuracy and requires less samples. These results show that neither differentiability nor even continuity are needed to work on a model combined with a deep learning perceptual front-end [29].

Finally, we show this approach can be used to learn preferences on an existing car configuration benchmark [13,14] where past configurations are available together with known manufacturing constraints. In this case, we observe that the learned preferences help to predict satisfactory configurations. The corresponding code will be made accessible from the TOULBAR2 distribution (`https://github.com/toulbar2/toulbar2`, under an MIT licence).

## 2 Background

Our approach is based on Graphical Models [10], a family of mathematical models that has been used in several areas of computer science, artificial intelligence,

physics and statistics. The main idea of Graphical Models is to describe a function of many variables as the combination of several simple functions. "Simple" here means that there is a concise description of the function in a chosen language of functions. Graphical Models have been used to describe Boolean or numerical functions depending on continuous (as in Gaussian Graphical Models [8]) or discrete variables (as in Constraint Networks [32] or propositional logic).

- On the logical side, Constraint networks define a global truth value function as the logical conjunction of small functions described by tables (Boolean tensors), possibly extended with so-called global constraints in Constraint Programming [32].
- Similarly, discrete Markov Random Fields describe a probability distribution as the normalization of the product of small non negative functions described as tables (non negative real tensors), possibly extended with higher-order functions (similar to global cost functions [1]).

In the rest of the paper we use capitals $X, Y, Z, \ldots$ to denote variables. The domain of a variable will be denoted as $D^X$ for variable $X$. The actual elements of these domains, values, will be denoted as $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{g}, \mathtt{r}, \mathtt{t}, \mathtt{1} \ldots$ and an unknown value as $u, v, w, x, y, z \ldots$ Sequence of variables or unknown values will be denoted in bold, respectively as $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}, \ldots$ and $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \ldots$. The Cartesian product of the domains of a sequence of variables $\boldsymbol{X}$ will be denoted as $\Pi^{\boldsymbol{X}}$. An element of $\Pi^{\boldsymbol{X}}$ is a tuple or assignment $\boldsymbol{u_X}$ of the variables in $\boldsymbol{X}$. Finally, the projection of the tuple $\boldsymbol{u_X}$ on $\boldsymbol{Y} \subseteq \boldsymbol{X}$ is the sequence of values of $\boldsymbol{Y}$ in $\boldsymbol{u_X}$ and is denoted as $\boldsymbol{u_X}[\boldsymbol{Y}]$. For a given sequence of numbers $\boldsymbol{x} = (x_1, \ldots, x_n)$, its soft-max is $\log(\sum_{x_i \in \boldsymbol{x}} \exp(x_i))$ and its soft-min $-\log(\sum_{x_i \in \boldsymbol{x}} \exp(-x_i))$. Soft-max provides a usual smooth approximation to the maximum function (as does soft-min for the minimum).

We rely on two closely related types of Graphical Models. Cost Function Networks are an extension of Constraint Networks where constraints (Boolean functions that can be satisfied or not), are replaced by bounded integer functions that are summed together to describe a joint bounded numerical function.

**Definition 1 (Cost Function Networks (CFN)).** *A CFN $\mathcal{C} = \langle \boldsymbol{V}, \boldsymbol{C}, k \rangle$ is defined by:*

- *a sequence of $n$ variables $\boldsymbol{V}$, each with a domain of cardinality at most $d$.*
- *a set $\boldsymbol{C}$ of $e$ cost functions.*
- *each cost function $c_{\boldsymbol{S}} \in \boldsymbol{C}$ is a function from $D_{\boldsymbol{S}} \to \bar{\mathbb{Z}}_k$, the set of all integers less than or equal to a given $k \in \bar{\mathbb{Z}} = \mathbb{Z} \cup \{\infty\}$.*

*The CFN $\langle \boldsymbol{V}, \boldsymbol{C}, k \rangle$ defines a joint function $C_{\mathcal{M}}(\boldsymbol{v}) = \bigoplus_{c_{\boldsymbol{S}} \in \Phi}^{k} c_{\boldsymbol{S}}(\boldsymbol{v}[\boldsymbol{S}])$ where $a \oplus b = \min(a + b, k)$, the bounded addition.*

Computing the minimum cost assignment of a CFN is the *Weighted Constraint Satisfaction Problem* (WCSP). Thanks to the upper bound $k$, CFNs are very flexible. For $k = 1$, CFNs are Constraint Networks. Finite values of $k$ capture

situations in which an upper bound is known (e.g., the maximum cost that can be spent in a design).

Cost Function Networks are tightly linked to a family of stochastic Graphical Models known as Markov Random Fields:

**Definition 2 (Markov Random Field (MRF)).** *An MRF* $\mathcal{M} = \langle \boldsymbol{V}, \Phi \rangle$ *is defined by:*

- *a sequence of* $n$ *variables* $\boldsymbol{V}$*, each with a domain of cardinality at most* $d$*.*
- *a set* $\Phi$ *of* $e$ *functions (or factors).*
- *each function* $\varphi_{\boldsymbol{S}} \in \Phi$ *is a function from* $\Pi^{\boldsymbol{S}} \to \mathbb{R}^+$*.* $\boldsymbol{S}$ *is called the scope of the function and* $|\boldsymbol{S}|$ *its arity.*

*The MRF* $\langle \boldsymbol{V}, \Phi \rangle$ *defines a joint function* $\Phi_{\mathcal{M}}(\boldsymbol{v}) = \prod_{\varphi_{\boldsymbol{S}} \in \Phi} \varphi_{\boldsymbol{S}}(\boldsymbol{v}[\boldsymbol{S}])$ *and a probability distribution defined as* $P_{\mathcal{M}}(\boldsymbol{V}) \propto \Phi_{\mathcal{M}}(\boldsymbol{V})$*.*

Computing the probability $P_{\mathcal{M}}(\cdot)$ requires to compute a normalization constant, denoted as $Z_{\mathcal{M}}$, a #-P complete problem. For a given MRF $\mathcal{M}$, finding an assignment $\boldsymbol{v}$ that maximizes the probability $P_{\mathcal{M}}(\boldsymbol{v})$ can however be directly solved by optimizing the joint function $\Phi_{\mathcal{M}}(\cdot)$, without knowing $Z_{\mathcal{M}}$ and is decision NP-complete.

The connection between CFNs and MRF is simple: in a CFN with no upper bound ($k = \infty$), $\oplus$ is just the usual addition. In this case, CFNs are isomorphic to Markov Random Fields through a $\exp(-x)$ transform and its inverse $-\log(x)$ transform, up to some adjustable fixed precision. These operations map addition into product (and vice-versa[1]). For a given MRF $\mathcal{M}$, we denote by $\mathcal{M}^{\ell}$ its corresponding CFN, obtained by applying a $-\log(\cdot)$ transform to all functions.

In CSPs, CFNs, and MRFs, a usual choice is to represent functions by tables/tensors. When domains are Boolean, the language of (weighted) clauses can also be used. We restrict ourselves here to pairwise tensors where each function is determined by an $O(d^2)$ table of costs (or parameters). Then a pairwise graphical model becomes fully defined by the contents of its $O(\frac{n(n-1)}{2})$ cost tables (if no function exists between a given pair of variables, it can be represented as a cost function with constant cost). Extensions to larger arities and global cost functions are not considered here and define non trivial extensions for the future.

## 3 Learning CFN from data

In many decision problems, a fraction of the description of the real problem is impossible to model because this information is missing or is too complex to represent. In the extreme, one may want to directly learn a complete CFN from data (a special case of which is Max-SAT [23]).

[1] This log representation is often using in MRFs and the co-domain of factors is called "energy".

**Definition 3 (Learning CFN).** *Given a set of variables $\boldsymbol{X}$, and examples $\boldsymbol{E}$ sampled i.i.d. from an unknown joint distribution of high-quality solutions, find a CFN $\mathcal{C}$ that can be solved to produce high-quality solutions.*

Thanks to their isomorphism with Markov Random Fields, CFN can actually be learned in this setting using a probabilistic criterion. Several approaches exist to estimate the set of functions of an MRF from an i.i.d sample but a good fit with the definition above is offered by maximum log-likelihood approaches that learn a model $\mathcal{M}$ that maximizes the probability of the observed sample. Indeed, the likelihood of a sample $\boldsymbol{E}$ of i.i.d. assignments under a given MRF $\mathcal{M}$ is the product of the probabilities of all $\boldsymbol{v} \in \boldsymbol{E}$. Its logarithm is:

$$\begin{aligned}\mathcal{L}(\mathcal{M}, \boldsymbol{E}) &= \textstyle\sum_{\boldsymbol{v}\in\boldsymbol{E}} \log(P(\boldsymbol{v})) \\ &= \textstyle\sum_{\boldsymbol{v}\in\boldsymbol{E}} \log(\Phi_{\mathcal{M}}(\boldsymbol{v})) - \log(Z_{\mathcal{M}}) \\ &= \textstyle\sum_{\boldsymbol{v}\in\boldsymbol{E}} (-C_{\mathcal{M}^\ell}(\boldsymbol{v})) - \log(\sum_{\boldsymbol{v}\in\Pi^{\boldsymbol{V}}} \exp(-C_{\mathcal{M}^\ell}(\boldsymbol{v})))\end{aligned}$$

Maximizing the log-likelihood therefore identifies weights in all possible pairwise tensors that simultaneously minimize the average cost of the observed high-quality solutions and maximize the soft-min of the costs of *all* possible assignments, a criterion which fits our optimization objective above very well, independently of its probabilistic interpretation. For a Graphical Model of $n$ variables and maximum domain size $d$, there are $O(n^2d^2)$ weights to optimize.

### 3.1 Regularized approximate log-likelihood GM estimation

In practice computing the partition function $Z_{\mathcal{M}}$ is #P-hard. Existing algorithms therefore optimize a simplified form of the likelihood which either relies on local normalization constants (pseudo-likelihood [4]) or a concave upper-bound of the log-partition function [30]. Maximum likelihood estimators benefits from attractive asymptotic properties, being statistically consistent (the model learned converges to true values as the sample size tends to infinity) [30,17]. On small samples however, these approaches may overfit and the log-likelihood is regularized by including the norm of the parameters learned as a penalty. Typical norms include the $L_2$ norm (the Euclidian norm), the $L_1$ norm (or Lasso penalty, the sum of the absolute values of all parameters learned) or the $L_1/L_2$ norm (or Group Lasso, that evaluates each function using the $L_2$ norm and combines them using the $L_1$ norm). Given an i.i.d sample $\boldsymbol{E}$ of assignments, the regularized log-likelihood of an MRF $\mathcal{M}$ is defined as:

$$\mathcal{R}(\mathcal{M}, \boldsymbol{E}) = \mathcal{L}(\mathcal{M}, \boldsymbol{E}) - \lambda \cdot ||\Phi||$$

where $||\Phi||$ denotes the norm of all the parameters used in the tensors in $\Phi$ and $\lambda$ is a positive number that needs to be fixed. The Lasso norms ($L_1$ or $L_1/L_2$) bias the criteria to favor functions that take a zero value. This has several positive effects: a function with a table full of zeros does not contribute to the value of the joint function and can be removed, allowing to estimate parameters and scopes simultaneously. Our experiments also show that Lasso regularization can

effectively cancel the unavoidable sampling noise present in the finite learning set that otherwise leads to the estimation of a Graphical Model that contains a fraction of random cost functions that are very hard to optimize exactly.

To solve this problem, we rely on a recently proposed scalable (in $O(n^3d^3)$ for pairwise tensors) regularized maximum log-likelihood estimation algorithm, PE_MRF [30], that exploits the ADMM (Alternating Direction Multiplier Method) algorithm for optimization [9]. The algorithm has been designed to learn a GM from a set of solution samples but is actually immediately capable of learning using probabilistic input which will prove very useful in the most intense interaction with deep learning systems later. This algorithm can also learn mixed graphical models with both discrete and continuous variables which can be convenient if the learning set includes not only decision variables but also contextual continuous observations that will also be available when solving (even if we don't explore this capacity further in this paper). This regularized approximate log-likelihood approach using the $L_1/L_2$ norm has been shown to be "sparsistent": as the size of the learning set tends to infinity, the probability of finding the exact graph structure tends to 1 [30], a reassuring asymptotic result, even if our target is to learn a solver, not to estimate a graph structure.

Although the ADMM algorithm is a black box optimization algorithm, it is useful to understand how it works. ADMM is well-suited to optimize convex functions that are sums of terms. Using a Dual Decomposition principle, every optimization variable $c_{ij}(a, b)$ (the cost of the pair $(a, b)$ in the function $c_{i,j}(\cdot, \cdot)$ of the GM to learn) is duplicated into a copy $c'_{ij}(a, b)$ and the two parameters linked by an equality constraint. At each iteration, the log-likelihood is incrementally optimized on the $c$ variables while the regularization penalty is incrementally optimized on the $c'$ variables. The satisfaction of the equality constraints is delegated to an Augmented Lagrangian approach that penalizes the violation of constraints [9]. As the algorithm iterates, it constantly provides two estimates of the parameters, each defining a CFN. Upon convergence, the two copies are almost but not strictly identical. In practice, it is preferable to use the $c'$ copies which optimize for regularization and contain exact zero. This is crucial for exact WCSP solvers that otherwise spend a gigantic optimization effort optimizing tiny costs often reflecting uninformative sampling noise.

### 3.2 Setting the regularization parameter

The determination of a suitable value of $\lambda$ is essential for proper prediction. Existing approaches to tune this parameter in Machine Learning focus in recovering the unknown graph structure (which cannot be achieved using pseudo-likelihood in the presence of infinite costs [36]).

However, recovering the true graph is not our target and, similarly to what has been observed in the "Smart 'Predict and Optimize'" framework [11], we observed that taking into account the exact prediction objective does help. We therefore use en empirical risk (or error) minimization (ERM) approach. This approach is central in the recent HASSLE Partial Weighted Max-SAT algorithm [23] which also proves that Max-$p$-SAT and CFN models are Probably

Approximately Correct(ly) (PAC [35]) learnable by ERM. Using a solution $\boldsymbol{s}$ extracted from a validation set of high-quality solutions (ideally distinct from the training set used for PE_MRF), we assign a fraction of all variables in the learned CFN model with their value in $\boldsymbol{s}$ and ask a WCSP solver to optimize the remaining variables. The solution obtained can be correct (or close to $\boldsymbol{s}$ according to an application-specific distance that can default to the Hamming distance) or not. We use a value of $\lambda$ that minimizes the fraction of non-satisfactory assignments.

Optimizing $\lambda$ in this way requires the repeated resolution of a decision NP-complete problem on the validation set. This is a serious issue even on small problems because the problems learned with very low values of $\lambda$ usually define dense CFNs with functions that overfit the learning set and capture the sampling noise in the learned cost functions. These random problems are extremely hard to solve in practice. While the use of polynomial time approximations defined by (linear or convex) relaxations has been used with success in related approaches [26], two different approaches can be used to mitigate this complexity. First, we can assign a larger fraction of each solution $\boldsymbol{s}$ in the validation set before solving. This reduces complexity exponentially. Each solution $\boldsymbol{s}$ in the validation set can be used with several partial assignments in order to cover all scopes. Second, we can relax the requirement for an optimal solution using either a bounded optimization effort (as captured by cpu-time or numbers of backtracks), or by requiring an approximate guarantee (using e.g. a weighted criterion [31]), to avoid spending time on the optimization of very noisy overfitted problems.

### 3.3 Cost function hardening

If needed, and if the training set is reliable (with deterministic 0/1 probabilities on observed values), a similar empirical approach can be used to harden cost functions into constraints. For every non zero cost in the CFN learned, one can simply test if the corresponding combination is observed in any of the training samples. If not, its cost is set to $k$ (the maximum forbidden cost). This may lead to a learned problem that removes more solutions than it should (assuming the true problem is known) but will never make the learned problem inconsistent. In essence, this process is similar to the empirical/experimental method used to learn the general laws of Physics, which slowly evolve as data accumulates.

### 3.4 Related approaches

Constraint Acquisition [7] learns Constraint Networks from exact positive or negative answers to (partial) membership queries [2]. We instead primarily try to learn a criterion that is not known to be a Boolean feasibility, using a fixed set of high-quality assignments (mostly because good – working – solutions are more often conserved than bad ones, as does Nature for proteins). We also allow these solutions to be only accessible through an imperfect perceptive layer.

Hassle [23] is a recent algorithm for learning Partial Weighted Max-$p$-SAT (PWMSAT) problems from contextual positive and negative examples using

empirical risk minimization. The learned Max-$p$-SAT examples can then be fed to any PWMSAT solver, possibly with additional hard and soft constraints, as in our case. The main strength of HASSLE is its ERM formulation that can decide, for every possible $p$-clause, which one needs to be hard, weighted or removed to make every sample optimal in the model. This MIP grows however very quickly with the sample size and $p$. The MILP approximation proposed is tested on problems that include at most 20 variables and 91 clauses. A direct encoding of the Sudoku problems would require 729 propositional variables and several thousands clauses (with 9-clauses). It relies on a NP-hard formulation of learning (which is costly, but should be beneficial on small samples).

The "Smart 'Predict then Optimize'"(SPO [11,26]) framework has several connections with our approach. Like the surrogate loss of SPO, the convex loss we use (the opposite of penalized log-likelihood and its non probabilistic interpretation), is statistically consistent but is best suited for Graphical Models. The empirical adjustment of $\lambda$ using an empirical approach that relies on the final discrete optimization method instead of a pure ML criteria, such as Akaike information content (AIC) or Bayesian Information Content (BIC), similarly adapts learning to the final target of actually solving the learned problem.

Recurrent Relational Neural Nets have been recently proposed as a "learning to reason" approach [29]. As in our case, they start from positive examples to later produce solutions. The RRN approach makes little assumptions on the pairwise functions to learn but directly exploits the graph structure of the problem that needs to be solved. On these edges, it learns pairwise "message passing" functions which are applied recurrently using an LSTM neural net [33]. These functions are then applied repeatedly in the prediction phase, similarly to what is done in Loopy Belief Propagation (and Arc Consistency). The resulting Neural Net is restricted to solving the problem it has been trained on and will not accept later side-constraints, something which is often desirable in practice.

SAT-net [37] is a Neural-net friendly approach using low-rank convex optimization to both optimize the parameters of a variant of Goemans and Williamson Max-2-SAT convex relaxation [18] and find good solutions using the associated randomized rounding approach. There is a likely similarity between the G&W relaxation (that SAT-net exploits) and the convex relaxations in PE_MRF (that we exploit), but in our case this relaxation is used only for learning, and is optimized by ADMM instead of coordinate descent. More importantly, we rely on a non-differentiable exact WCSP solver for prediction instead of the convex relaxation again. The WCSP solver provides adjustable optimization guarantees while the convex relaxation power is fixed. Furthermore, it is able to satisfy later added side-constraints, something which is impossible by solving G&W convex relaxation (but which would be feasible using a Max-SAT solver, something that has never been tested in our knowledge[2]).

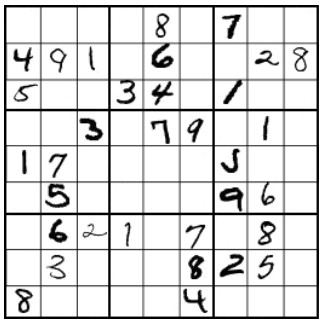
[2] The weights learned in the convex relaxation are floating point numbers. A precise integer approximation generates large integer costs which are usually not the sweet spot of the most efficient, core-based, Max-SAT solvers [28].

Probabilistic Soft Logic [3] is a related ML system which, as SAT-Net, exploits a convex relaxation for learning parameters and solving Graphical Models (using ADMM instead of coordinate descent). While it benefits from a high-level modeling language with first-order-like syntax, it has the same intrinsic limitation as SAT-Net: the convex relaxation has a fixed inference power and cannot provide guarantees that additional logical constraints will be satisfied.

## 4 Learning to solve the Sudoku

Neural Nets and differentiable approaches (such as RRN and SAT-Net) are now able to "learn to reason" from examples, providing the capacity to heuristically solve decision problems with little assumptions and from various inputs, including images. The Sudoku problem has been used as an exemplar of reasoning and we decided to apply our learning and reason architecture to the Sudoku problem, in an experimental setting that is comparable to those used by differentiable approaches in terms of assumptions and biases, also including situations where examples on which to learn require a perceptive layer.

The $n \times n$ Sudoku problem is defined over a grid of $n^2 \times n^2$ cells that each contain a number between 1 and $n^2$. This grid is subdivided in $n^2$ sub-grids of size $n \times n$. A solved Sudoku grid is such that the numbers in every row, column and $n \times n$ sub-grids are all different. Initially, a fraction of all cells is fixed to known values (or hints) and the NP-complete problem [38] is to find a completion of the hints that satisfies the constraints. The puzzle is usually played with $n = 3$. A typical grid, with handwritten hints from the MNIST dataset [24], is represented on the right. As all correct Sudoku puzzle grids, it has only one correct completion (a unique solution). It is known that a minimum of 17 hints is necessary to restrict the number of completions to just one [27]. Such minimal Sudoku problems define challenging puzzles for human beings. As the number of hints increases, the instances become easier and can be solved using simple logical inference rules. Hard or easy for humans, $3 \times 3$ instances can be solved easily by CP solvers, on any standard hardware.



We instead assume that we don't know much about the Sudoku, not even that it has logical rules. We instead consider that the completed grids capture the preferences of users and try to learn a CFN that captures these preferences and compare this with RRN and SAT-Net. It's not easy to compare language biases: RRN is informed with pairwise scopes, SAT-Net uses Max-SAT and we use pairwise finite costs CFNs. Max-SAT and pairwise numerical functions are both capable of representing the set of Sudoku solutions as optimal solutions. SAT-Net has, however, the attractive capacity of using latent variables.

SAT-Net relies on a dataset of 9,000 training + 1,000 test (hint,solution) pairs extracted from a popular Sudoku web site, with an a average of 36.2 hints per grid, defining easy problems. RRN relies on 180,000 training + 18,000 validation and 18,000 test pairs organized each in 18 sets of instances with hardness varying from 17 to 34 hints. We therefore used a variable fraction of the RRN training set for training and 1,024 validation samples for hyper-parameters tuning. For testing, we used all $18 \times 1,000$ RRN test samples as well as the SAT-nets test set for comparison. An Intel XeonE5-2687Wv4 3.00GHz server was used for all experiments. The absolute and relative convergence of ADMM in PE_MRF were both set to $10^{-3}$ and an L1-norm used. We used TOULBAR2 1.0.1 Python interface, representing floating point numbers with 6 decimals and with a number of backtracks limited to 50,000. The Python-implemented PE_MRF code used 72 seconds on average for one CFN estimation (with a maximum of 252 seconds on the largest 180,000 training set).

The empirical approach was used to fix the regularization hyper-parameter $\lambda$. We used TOULBAR2 to minimize the solution cost (in the backtracks limit) and kept the value of $\lambda$ that successively minimizes the fraction of incorrect grids, incorrect cells and TOULBAR2 cpu-time. The optimization of $\lambda$ (in a $10^{-2}$ to $10^{2}$ range explored on a logarithmic scale) took 95 minutes on average on one core (this could be trivially reduced with more cores). SAT-Net requires 172 minutes to train on a GTX 1080 Ti GPU on its training set [37]. We retrained RRNs on their training data on a GTX 2080 Ti GPU (with a batch size of 64): each epoch required 9 hours to run (hundreds of epochs are used by the authors [29]).

Using 180,000+18,000 training and validation samples, RRNs are able to correctly solve 96.7% problems of the hardest 17 hints problems using 64 "message passing" steps (after which it plateaus). Using 9,000 + 1,024 training and validation samples, our approach solves 100% of the same hard 17 hints problems.

Using just 9,000 samples, SAT-net is able to solve 98.3% of its test set (of easy problems). To solve 100% of this test set, 7,000 + 1.024 training and validation samples suffice for our architecture (on these problems, 994/1000 instances are solved backtrack-free, by preprocessing, the remaining problems requiring a total of 24 backtracks). Note however that the learned solver is able to solve only 58.2% of the hardest 17 hints problems. Clearly, problem hardness has to be taken in to account when comparing learned solvers. Figure 1 shows the fraction of correctly solved grids (left) as the sample size increases (performances beyond 13,000 samples are not shown and remain maximal).

The corresponding prediction cpu-times are given in Figure 1 (right). When training sets are small, the learned CFN models are dense. With 1,000 training samples, more than 1,700 functions are used while the original pairwise Sudoku formulation contains 810 functions. Because of this graph density (and their noisy contents), optimization is more difficult with small training sample sizes. As the training set size increases, the number of functions converges to 810 and resolution becomes easy: an optimal solution can be found and proved in sub-second time (with 9,000 samples, less than 0.3 seconds are needed on average for the hardest 17 hints Sudokus, and just 3ms for 34 hints problems). RRN's

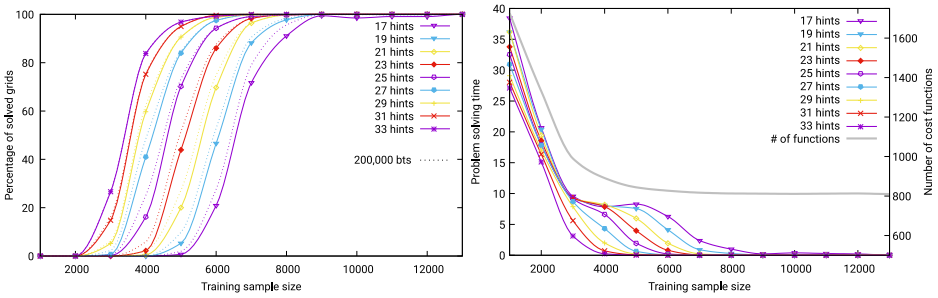



**Fig. 1.** Fraction of correctly solved problem (left, dotted lines correspond to a 200,000 backtracks limit), number of learned functions and per instance prediction cpu-times (right) for increasing sample sizes and problem hardness.

prediction time on a GTX 2080 Ti GPU was around 2 seconds for 64 steps. To see if more WCSP solving power could help improve these results, we moved the backtrack limit to 200,000 backtracks. This lead to minor improvements in precision as the dotted lines in Figure 1 show above, with essentially no progress in terms of accuracy on easy problems: a better loss function and a stronger learning optimization method would be needed here to make progress.

We observed that when the training set size reaches 13,000 samples, the learned CFNs become exact (the set of optimal solutions may be exact before this): they contain 810 cost functions with the exact expected scopes (involving pairs of variables inside a row, column or sub-grid only, although no grid layout information is available to PE_MRF) and contents (a "soft difference" function). So, it is guaranteed that, once domains are reduced by observed hints, a preferred (optimal) solution will be a perfect Sudoku solution. Empirical hardening (§ 3.3) of such a CFN therefore recovers the original pairwise formulation of the problem.

### 4.1 Learning and predicting from Sudoku images

One of the advantage of differentiable layers that "learn how to reason" is their capacity to integrate inside deep learning architectures. As an example, SAT-net [37] has been trained using hints provided as images with handwritten digits (an example of which appeared in a previous page). Each cell in this image can be decoded by LeNet [24], a convolutional neural net trained on the MNIST dataset with 99.2% precision. The predictions of LeNet are then fed into the SAT-Net layer for learning and prediction. As the authors of SAT-Net observe, the 99.2% precision of LeNet gives an upper bound on the maximum prediction precision: since the SAT-Net data-set has, on average, 36.2 hints per sample, there will be error(s) in the hints in 25.3% of cases, leading to a maximum accuracy of 74.7%.

We also used LeNet and transformed its confidence scores in a marginal unary cost function using soft-max, as is usual with neural net outputs. When a digit appears in a Sudoku image, this unary cost function is added to the learned model instead of assigning a value. This happens both during validation and

testing. Because solutions are available as images only, it becomes impossible to directly compare a predicted solution with the true (unknown) solution. We therefore apply LeNet to each cell of the solution image and use the value of the soft-max output of LeNet on the predicted digit as a score. A high score represents an unlikely digit for LeNet and we therefore select a $\lambda$ that produces the most likely solutions *i.e.*, which minimizes the sum of all such scores.

With hints and solutions provided as images during training, SAT-Net solves 63.2% of its test set using the same 9,000 samples. Going beyond SAT-Net, we used a more realistic setting where both hints and solutions are provided as imagesTo handle this situation, we exploited the fact that PE_MRF accepts as input *expectations* of sufficient statistics which can be produced from the marginal unary cost functions above using a $\exp(-x)$ transform. These marginal probabilities are used directly for computing expected numbers of values and pairs (the product of the two marginal probabilities $P(a) \times P(b)$ being used for pairs $(a, b)$). Using 8,000 + 1,024 training and validation samples, and a 200,00 backtracks limit, our hybrid architecture is able to solve 76.3% of all SAT-Net test problems. On the hardest 17 hints instances however, performance decreased to 61.8%. Obviously, hardening is of no use here.

## 5 Learning car configuration preferences

In this experiment, we illustrate the versatility of our approach by learning user preferences combined with logical information on a real configuration problem provided by Renault, a French car manufacturing company. A car configuration problem is defined by a set of variables, one for each type of option (engine, color, etc.). Domain values are possible options for each variable. Constraints describe manufacturing compatibilities between options.[3]

There are three datasets available, *small*, *medium*, and *big*, each one given in two files: a 1-year sales history of car configurations and a set of manufacturing constraints. The sales history products may or may not satisfy the constraints. We consider here only valid products. *medium* is a small urban car defining a toy example with 148 variables and 44 decision variables in the sales history,[4] mostly Boolean domains with a maximum size of 20 values, 173 constraints defined as tables with a maximum arity of 10, and 8,252 configurations consistent with the constraints in the history. *big* is a utility van with extensive product variability. It has 268 variables (87 decision variables in the sales history), 324 values at most per domain, 332 constraints with a maximum arity of 12, and 8,337 consistent configurations. We discarded the *small* instance as its sales history contains only 710 valid configurations. We counted $278,744$ (resp. $24,566,537,954,855,758,069,760 \approx 2^{74}$) feasible configurations

[3] See `https://www.irit.fr/~Helene.Fargier/BR4CP/benches.html`.
[4] We removed the first variable corresponding to the date of each sale product.

for *medium* (resp. *big*) in 0.1 (resp. 1.8) seconds on a 3.3GHz laptop.[5]. We used a 10-fold cross validation. The valid sales history was split into 10 folds so that all the identical car configurations were contained in the same fold. 9 folds were used as training set for learning user preferences and the last fold was used as test set for predicting user choices. This protocol was repeated 10 times.

We learn the user preferences based on the training set using PE_MRF with either $L_1$ or $L_1/L_2$ norm and a $\lambda$ parameter tuned using the StARS [25] algorithm among a logarithmic grid of 100 spaced values between $10^{-5}$ and $10^3$. We used the default value 0.05 for the threshold parameter $\beta$ in StARS and used subsamples of size $10\sqrt{n}$, where $n$ is the size of the training set, as advised in [25]. Mean $\lambda$ selected value was 34.6 (resp. 0.21) for *medium* (resp. *big*) using $L_1$ norm. The resulting learned CFN on decision variables had 312.9 (resp. 127.2) binary functions and 44 (resp. 87) unary functions.

Then, we test the learned CFN model combined with the manufacturing constraints[6] on the test set, using the protocol described in [13]. This protocol simulates an on-line configuration session with a user. For each test configuration $C$, we select a random variable ordering. Then, we predict the most-probable value for the next variable in the sequence, using the choices made by the user before in the sequence, the learned preferences, and the manufacturing constraints. 10 random variable orderings were considered for each test configuration. Instead of finding the most-probable value by discrete integration over the remaining variables (a marginal MAP inference task), we identified the most-probable valid configuration for all the variables compatible with the previous user choices and the constraints (a pure optimization Maximum A Posteriori – MAP – approximation of marginal MAP).[7] We compare the predicted value for the next variable with the one chosen in $C$ in order to compute a precision score.

We compared our method against a naive Bayesian network approach (called Naive Bayes) and an oracle method, as described in [12,14]. The structure of Naive Bayes is a tree rooted at the next variable with all the previously chosen variables as its sons. It makes the (unrealistic) assumption that all the leaf variables are independent knowing the root variable. The most probable value $v$ for the next variable $V$ given the assigned values $\boldsymbol{u}$ of $\boldsymbol{U}$ is easy to compute, $P(v|\boldsymbol{u}) \propto P(v) \prod_{X \in \boldsymbol{U}} P(\boldsymbol{u}[X]|v)$, based on precomputed priors $P(V)$ and conditional probability tables $P(U|V)$ for all pairs of variables $U, V$. The oracle method computes the posterior probability distribution of the next variable knowing the test dataset and the previous user choices. It uses the probability distribution estimated from this set and recommends, given the assigned values $\boldsymbol{u}$, the most probable value in the subset of products, in the test set, that respects $\boldsymbol{u}$. More precisely, for any value $v$ in the domain of the next variable to predict,

[5] Solution counting was done by Backtracking with Tree Decomposition algorithm [15] using *min-fill* heuristic implemented in TOULBAR2 v1.0.1 with options *-ub=1 -a -O=-3 -B=1 -hbfs: -nopre* Reported tree-width was 10 for *medium* and 12 for *big* instance.

[6] We ensure our CFN and the XCSP2.1 XML file for the constraints use the same variable domains with the same increasing value ordering.

[7] We used TOULBAR2 v1.0.1 with a limit of 50,000 backtracks and no preprocessing.

it estimates $P(v|\boldsymbol{u})$ as $\#(\boldsymbol{u}v)/\#(\boldsymbol{u})$. So, the oracle method is maximally fitted to the test set and its success rate is generally not attainable without having access to the test set (which is, obviously, not the case in practice). Its precision is not 100% since there is an intrinsic variability in the users [12].

The results for the different methods are given in Fig. 2, showing the average precision score and standard-deviation for varying number of hints. We report only results on the $L_1$ ($L_1/L_2$ gave the same precision scores). It took less than 1min. for *medium* (resp. 2min. for *big*) to learn preferences and collect all the 36,124 (resp. 73,428) precision scores for a single fold of cross-validation.[8] The maximum number of backtracks was less than $1,000$, much less than its limit.

The average precision was 93.41% (resp. 94.41% for *big*) compared to 92.08% (resp. 92.31%) for Naive Bayes and 97.10% (resp. 97.35%) for the oracle, showing the practical interest of our approach on real recommendation datasets, providing high precision values in a reasonable amount of time. However, when the number of hints is small, Naive Bayes performed slightly better (for *big*) than our approach, possibly due to the MAP approximation.

Moreover, we investigated the impact of removing either the preferences, partially by just removing learned binary functions and keeping learned unary terms (*L1 without learnt binaries* curve in Fig. 2), or removing all learned functions (*Only constraints*), or removing the manufacturing constraints (*L1 without constraints*). All these removals had a negative impact, showing the interest of combining preferences and logical knowledge. Other approaches have been developed to take into account constraints in recommendation systems, such as constraint propagation [5,6] or compilation techniques [19]. However, they remain separated from the preference model, whereas our approach exploits two CFNs (learned and mandatory constraint networks) on the same decision variables. We leave a full comparison of the MAP approximation with exact or approximate inference for predicting the next variable value as a future work.

## Conclusion

In this paper, we show that an hybrid architecture combining differentiable and non differentiable technologies from Graphical Model learning (ADMM convex optimization embodied in the PE_MRF algorithm) and solving (using an anytime Weighted CSP solver with adjustable guarantees) can provide excellent empirical performances, often outperforming recent neural net friendly approaches, with comparable biases. Purely differentiable approaches have the nice property to be directly usable inside more complex deep Learning architectures, often allowing a streamlined learning and predict architecture. By accepting numerical input in its learning component and integer costs in its prediction component, our architecture has the capacity of exploiting neural nets output at the very minor cost of a less streamlined but perfectly workable learning and solving process.

[8] We implemented an incremental version of the TOULBAR2 solving procedure using its Python interface in order to load the problem and preprocess it only once.

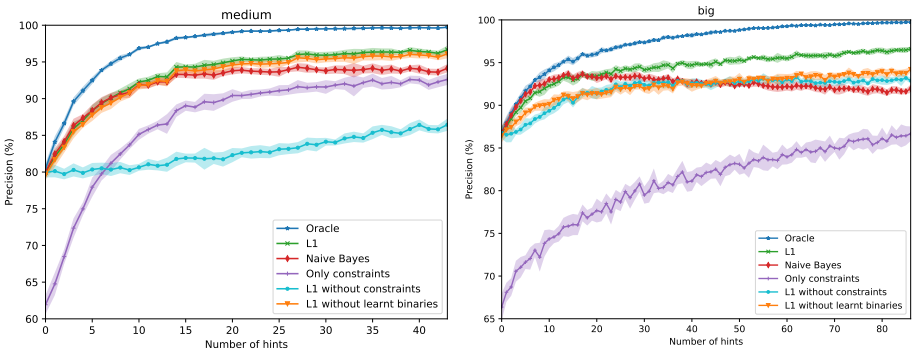



**Fig. 2.** Precision for the next query variable given a number of hints (on previously randomly-selected query variables).

Instead of relying on a polytime bounded inference power, as those offered by, e.g., message-passing or convex relaxations, it offers more powerful inference, associated to higher computational costs that can however be easily controlled either in terms of maximum computational effort or bounded guarantees. Because they provide strong polynomial time continuous approximations of discrete models such as Max-2SAT, convex relaxations have been repeatedly used as an ideal articulation point between learning and solving discrete/logical models. If the Unique Game Conjecture [21,22] holds, the most promising path for improvement seems to go beyond P and convex relaxations and use NP-complete formulations for solving and learning. This makes powerful anytime NP-hard numerical MIP, WCSP and PW-MaxSAT solvers of prime interest to make progress in this quest.

As an amusing yet puzzling coincidence, we observe that our hybrid approach is consistent with the dichotomy between the Systems 1 and 2 described in "Thinking Fast and Slow" for human cognitive limitations [20]. Beyond this coincidence; a more practical advantage of our hybrid approach is that it offers a decipherable output, that can be scrutinized to extract logical rules if they empirically reliably predict solutions but also directly used to enhance existing models that may contain mandatory constraints, as is often the case in design problems.

## Acknowledgements

We thanks the GenoToul (Toulouse, France) Bioinformatics and IFB Core (Evry, France) platforms for their computational support. We also thanks the reviewers for their critics: the paper did improve, we think. This work has been supported by the French ANR through grants ANR-16-CE40-0028 and ANR-19-PI3A-0004.

## References


1. Allouche, D., Bessière, C., Boizumault, P., de Givry, S., Gutierrez, P., Lee, J.H., Leung, K.L., Loudni, S., Métivier, J.P., Schiex, T., Wu, Y.: Tractability-preserving transformations of global cost functions. Artificial Intelligence **238**, 166–189 (2016)
2. Angluin, D.: Queries and concept learning. Machine learning **2**(4), 319–342 (1988)
3. Bach, S.H., Broecheler, M., Huang, B., Getoor, L.: Hinge-loss markov random fields and probabilistic soft logic. The Journal of Machine Learning Research **18**(1), 3846–3912 (2017)
4. Besag, J.: Efficiency of pseudolikelihood estimation for simple gaussian fields. Biometrika pp. 616–618 (1977)
5. Bessiere, C., Fargier, H., Lecoutre, C.: Global inverse consistency for interactive constraint satisfaction. In: Schulte, C. (ed.) Proc. of CP-13. pp. 159–174. Cork, Ireland (2013)
6. Bessiere, C., Fargier, H., Lecoutre, C.: Computing and restoring global inverse consistency in interactive constraint satisfaction. Artificial Intelligence **241**, 153 – 169 (2016)
7. Bessiere, C., Koriche, F., Lazaar, N., O'Sullivan, B.: Constraint acquisition. Artificial Intelligence **244**, 315–342 (2017)
8. Bishop, C.M.: Pattern recognition and machine learning, 5th Edition. Information science and statistics, Springer (2007), `http://www.worldcat.org/oclc/71008143`
9. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine learning **3**(1), 1–122 (2011)
10. Cooper, M., de Givry, S., Schiex, T.: Graphical models: Queries, complexity, algorithms. Leibniz International Proceedings in Informatics **154**, 4–1 (2020)
11. Elmachtoub, A.N., Grigas, P.: Smart" predict, then optimize". arXiv preprint arXiv:1710.08005 (2017)
12. Fargier, H., Gimenez, P., Mengin, J.: Recommendation for product configuration: an experimental evaluation. In: 18th International Configuration Workshop at CP-16. p. 8 p. Toulouse, France (2016)
13. Fargier, H., Gimenez, P., Mengin, J.: Learning lexicographic preference trees from positive examples. In: Proc. of AAAI-18. pp. 2959–2966. New Orleans, Louisiana, USA (2018)
14. Fargier, H., Gimenez, P.F., Mengin, J.: Experimental evaluation of three value recommendation methods in interactive configuration. Journal of Universal Computer Science **26**(3), 318–342 (2020)
15. Favier, A., de Givry, S., Jégou, P.: Exploiting problem structure for solution counting. In: Proc. of CP-09. pp. 335–343. Lisbon, Portugal (2009)
16. Freuder, E.C.: Progress towards the holy grail. Constraints **23**(2), 158–171 (2018)
17. Geman, S., Graffigne, C.: Markov random field image models and their applications to computer vision. In: Proceedings of the international congress of mathematicians. vol. 1, p. 2. Berkeley, CA (1986)
18. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM (JACM) **42**(6), 1115–1145 (1995)
19. Hadžic, T., Wasowski, A., Andersen, H.R.: Techniques for efficient interactive configuration of distribution networks. In: Proc. of IJCAI-07. pp. 100–105. Hyderabad, India (2007)
20. Kahneman, D.: Thinking, fast and slow. Macmillan (2011)

21. Khot, S.: On the power of unique 2-prover 1-round games. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. pp. 767–775 (2002)
22. Klarreich, E.: Approximately hard: The unique games conjecture. Simons foundation (2011)
23. Kumar, M., Kolb, S., Teso, S., De Raedt, L.: Learning MAX-SAT from contextual examples for combinatorial optimisation. In: Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence. AAAI (2020)
24. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
25. Liu, H., Roeder, K., Wasserman, L.: Stability approach to regularization selection (StARS) for high dimensional graphical models. In: Proceedings of Advances in Neural Information Processing Systems (NIPS 2010). vol. 24, pp. 1432–1440 (2010)
26. Mandi, J., Demirović, E., Stuckey, P., Guns, T., et al.: Smart predict-and-optimize for hard combinatorial optimization problems. In: Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence. AAAI (2020)
27. McGuire, G., Tugemann, B., Civario, G.: There is no 16-clue sudoku: Solving the sudoku minimum number of clues problem via hitting set enumeration. Experimental Mathematics **23**(2), 190–217 (2014)
28. Morgado, A., Heras, F., Liffiton, M., Planes, J., Marques-Silva, J.: Iterative and core-guided MaxSAT solving: A survey and assessment. Constraints **18**(4), 478–534 (2013)
29. Palm, R.B., Paquet, U., Winther, O.: Recurrent relational networks. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada. pp. 3372–3382 (2018)
30. Park, Y., Hallac, D., Boyd, S., Leskovec, J.: Learning the network structure of heterogeneous data via pairwise exponential Markov random fields. Proceedings of machine learning research **54**, 1302 (2017)
31. Pohl, I.: Heuristic search viewed as path finding in a graph. Artificial intelligence **1**(3-4), 193–204 (1970)
32. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier (2006)
33. Schmidhuber, J., Hochreiter, S.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997)
34. Simoncini, D., Allouche, D., de Givry, S., Delmas, C., Barbe, S., Schiex, T.: Guaranteed discrete energy optimization on large protein design problems. Journal of chemical theory and computation **11**(12), 5980–5989 (2015)
35. Valiant, L.G.: A theory of the learnable. Communications of the ACM **27**(11), 1134–1142 (1984)
36. Vuffray, M., Misra, S., Lokhov, A., Chertkov, M.: Interaction screening: Efficient and sample-optimal learning of Ising models. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29, pp. 2595–2603. Curran Associates, Inc. (2016)
37. Wang, P., Donti, P.L., Wilder, B., Kolter, J.Z.: Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA. Proceedings of Machine Learning Research, vol. 97, pp. 6545–6554. PMLR (2019)

38. Yato, T., Seta, T.: Complexity and completeness of finding another solution and its application to puzzles. IEICE transactions on fundamentals of electronics, communications and computer sciences **86**(5), 1052–1060 (2003)